307307 BI Methods

Introduction to Retrieval Augmented Generation (RAG) in KNIME

https://www.knime.com/events/ai-chatbots-rag-governance-data-workflows-course

Knowledge Limitations in LLMs

- LLMs are trained on general, static datasets, meaning their knowledge is fixed at the point of training.
- As a result, their responses are limited to what they've "seen" during that training phase.
- This creates limitations when LLMs are asked about:
  - Recent events
  - Proprietary or internal data
  - Highly domain-specific information

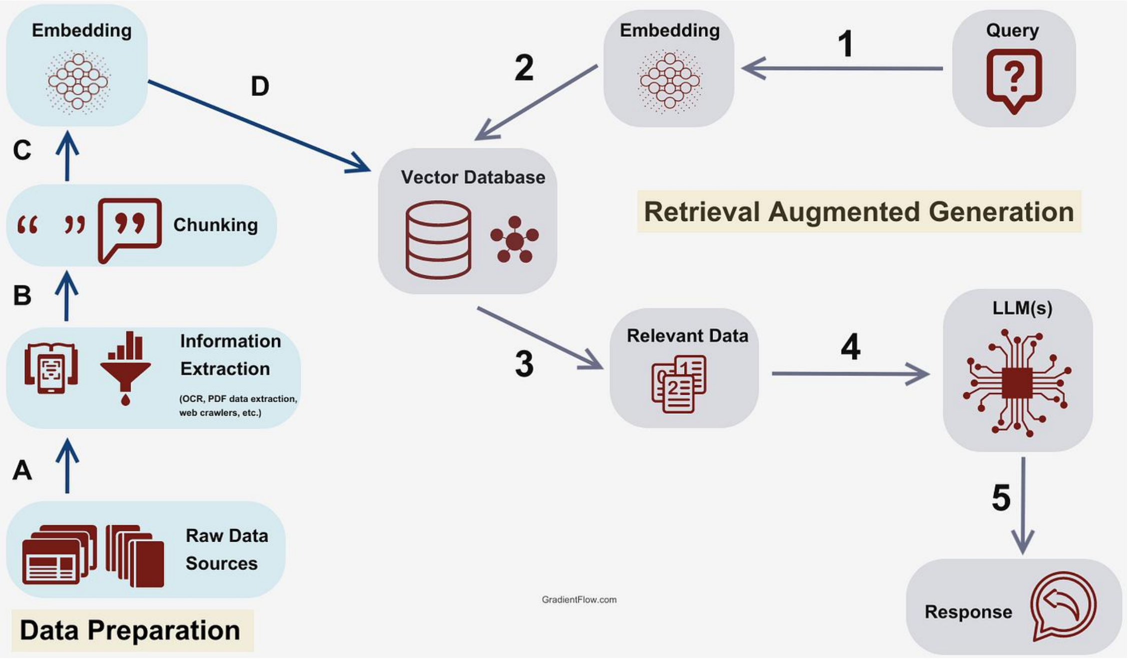
In these cases, LLMs may produce hallucinations (responses that sound plausible but are incorrect or misleading).

Knowledge Limitations in LLMs

- Tools like ChatGPT can overcome some of these limitations because they can access web search in real time.
- But when you access LLMs programmatically, via API or local models, e.g., in a KNIME workflow, they do not have access to external or updated data.
- To enhance the accuracy and relevance of LLM outputs, several approaches can be used:
    1. Retrieval-Augmented Generation (RAG)
    2. Fine-tuning

Retrieval-Augmented Generation (RAG)

- RAG stands for Retrieval-Augmented Generation.
- It is one of the simplest and most cost-effective techniques to overcome the limitations of LLMs' fixed knowledge.
- Rather than retraining the model, RAG works by augmenting the prompt with additional information that are automatically retrieved from an external knowledge base, based on the user's query.
- No retraining or fine-tuning is happening in RAG. The original LLM remains unchanged.
- At a high level, a RAG pipeline consists of these main elements:
  1. **Retrieval**. Identify and retrieve relevant information from a knowledge base based on the input query.
  2. **Augmentation**. Augment the original query or prompt with the retrieved information. This provides the model with additional context to better understand the task.
  3. **Generation**. Generate a more accurate and context-aware response using the augmented prompt.



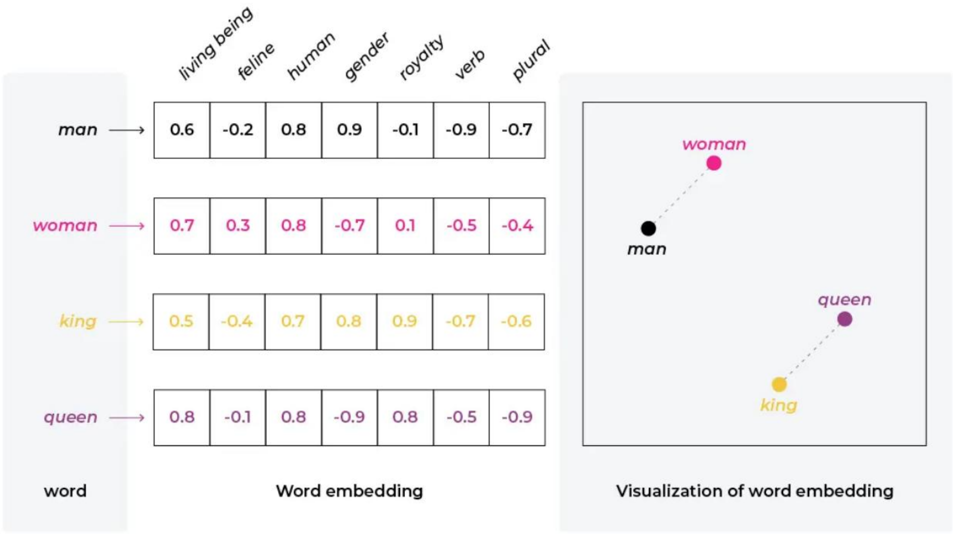
General Overview of the RAG Process

Word Embeddings and Vector Stores

- To understand how RAG works, we first need to understand two concepts essential for automated retrieval of textual information: **Embeddings and Vector Stores**.
- In RAG, our main goal is to *automatically* retrieve the most relevant pieces of information from a potentially large and unstructured free-text knowledge base.
- Why not just pass the entire knowledge base in the prompt to the LLM?
  - It might not fit within the LLM's context length.
  - It can increase computational costs.
  - It may make the prompt noisier and reduce response quality.
- So instead, the system needs to search and select only the most relevant snippets of information - automatically.

Word Embeddings and Vector Stores

- For an automated search of relevant information, the free-text data needs to be converted into a numerical form that can be compared. That form is embeddings.
- **Embeddings are** high-dimensional vector representations of text, where the position of each vector reflects the semantic meaning of the text it represents.
- Similar meanings are represented by similar vectors.



- An **Embedding Model** processes each chunk of our knowledge base and converts it into a high-dimensional vector. These vectors are positioned in a semantic vector space such that:
  - **Similar content** is placed **closer together**,
  - **Dissimilar content** is placed **further apart**.
- Once our knowledge base is embedded, the vectors need to be stored in a way that allows for efficient retrieval. That's the job of a **Vector Store**.
- A Vector Store is a database that stores embeddings and supports similarity search using vector distance metrics.

Vector Stores

What is a vector?

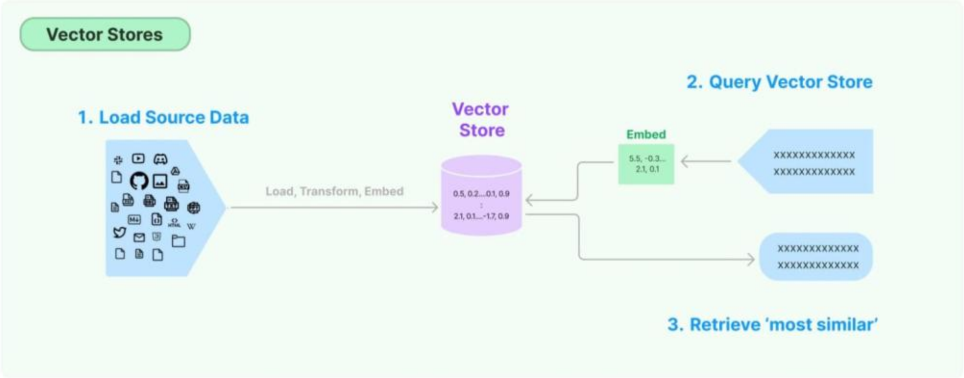
- A vector is an array of numbers that represents data (text, image, audio) in a high-dimensional space.
- When generated from text, these are called **embeddings**, which capture semantic meaning.

What is a vector store?

- A specialized database for storing and searching embeddings.
- Unlike relational databases (rows and columns), vector stores group embeddings by **similarity**.

Why are they important?

- Enable **fast similarity search**: find the most relevant documents for a query.
- Essential for RAG and AI-powered applications, since they provide the LLM with focused, relevant context instead of raw, full data.
- Allow systems to scale to very large knowledge bases without exceeding LLM context limits.



https://www.knime.com/blog/4-levels-llm-customization

How it works (simplified)

1. Data (text, image, etc.) → converted into embeddings using a certain LLM Model.
2. Embeddings are stored in a vector database.
3. User query → also embedded.
4. Vector store retrieves the **most similar documents**.
5. Relevant snippets are passed to the LLM to generate the final response.
6. **Examples of vector stores**: Chroma, FAISS, Pinecone, Weaviate.

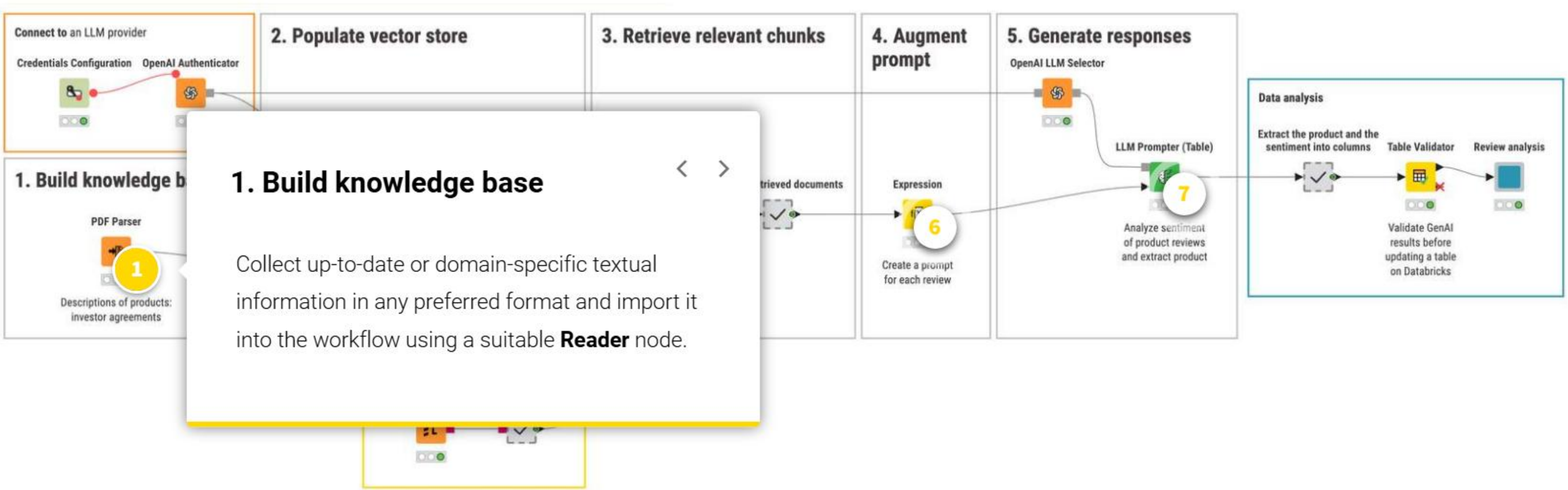
RAG in KNIME

KNIME provides all necessary components to build a complete RAG pipeline, including the ability to:

- Import documents in various formats from your knowledge base,
- Connect to embedding models, whether local or API-based,
- Retrieve relevant information from the vector stores, augment prompt, and generate the response.

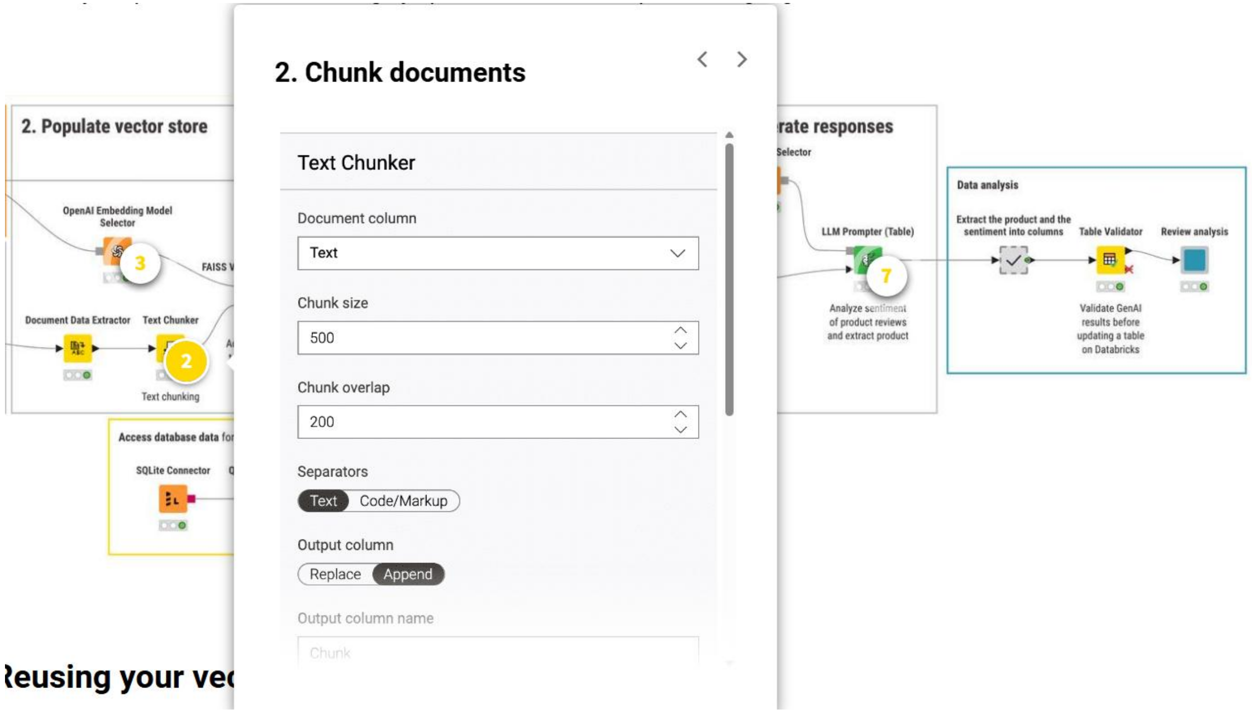
1- Collect Relevant Data

Collect up-to-date or domain-specific textual information in any preferred format and import it into the workflow using a suitable **Reader** node.



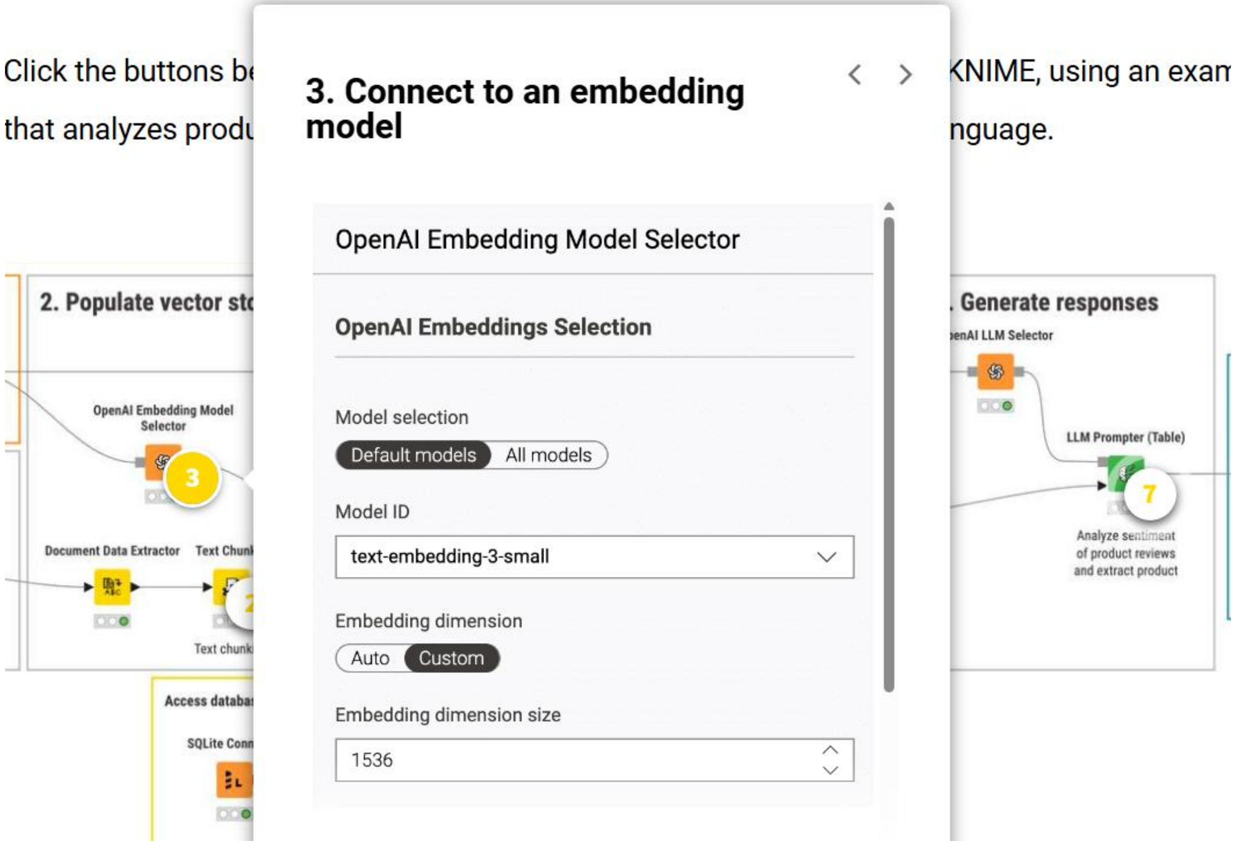
2- Chunk Documents

- If the knowledge base is a large document, split it into meaningful chunks for retrieval (not too short, not too long), taking into account the LLM's context window.
- We can use nodes such as Text Chunker or Sentence Extractor for this step.
- We can use the Text Chunker node to create chunks automatically while keeping semantic relations and considering formatting language syntax.



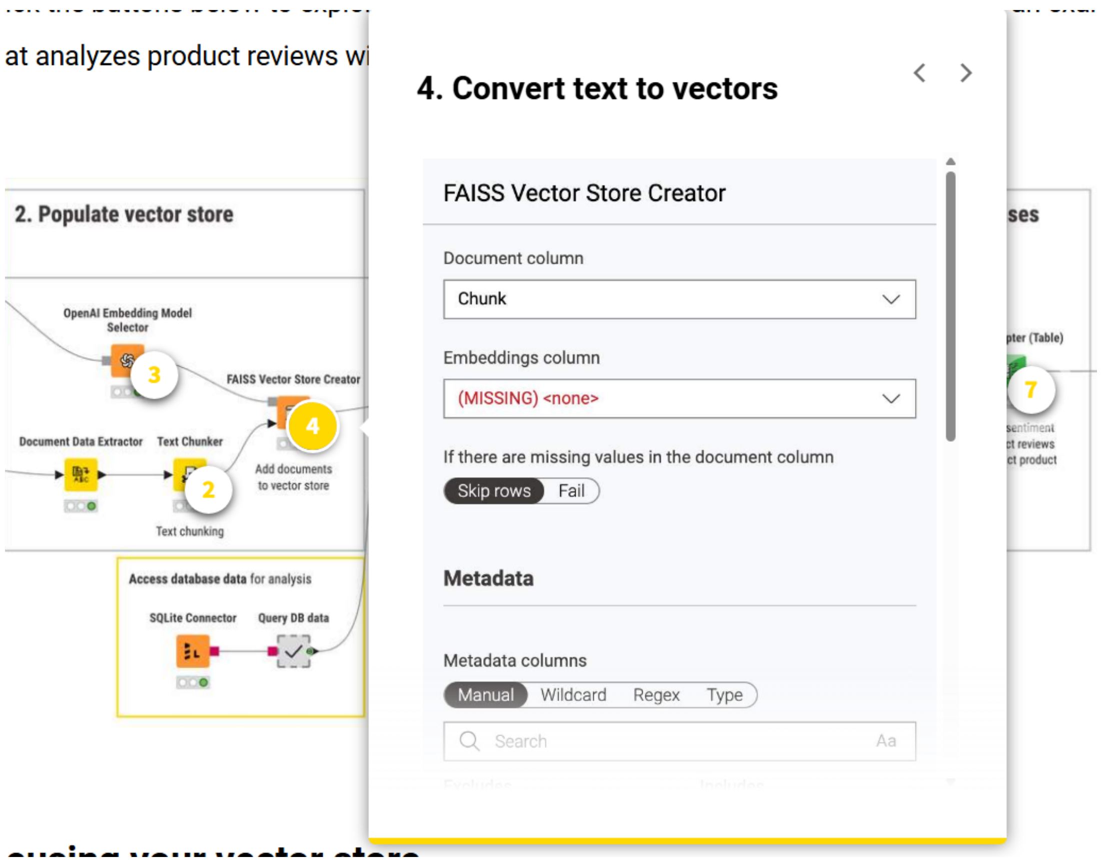
3- Connect to an Embedding Model

- Connect to our LLM provider and select an embedding model of your choice using a suitable **Embedding Model Selector** node.
- Alternatively, connect to a local embedding model using the **GPT4All Embedding Model Selector**.



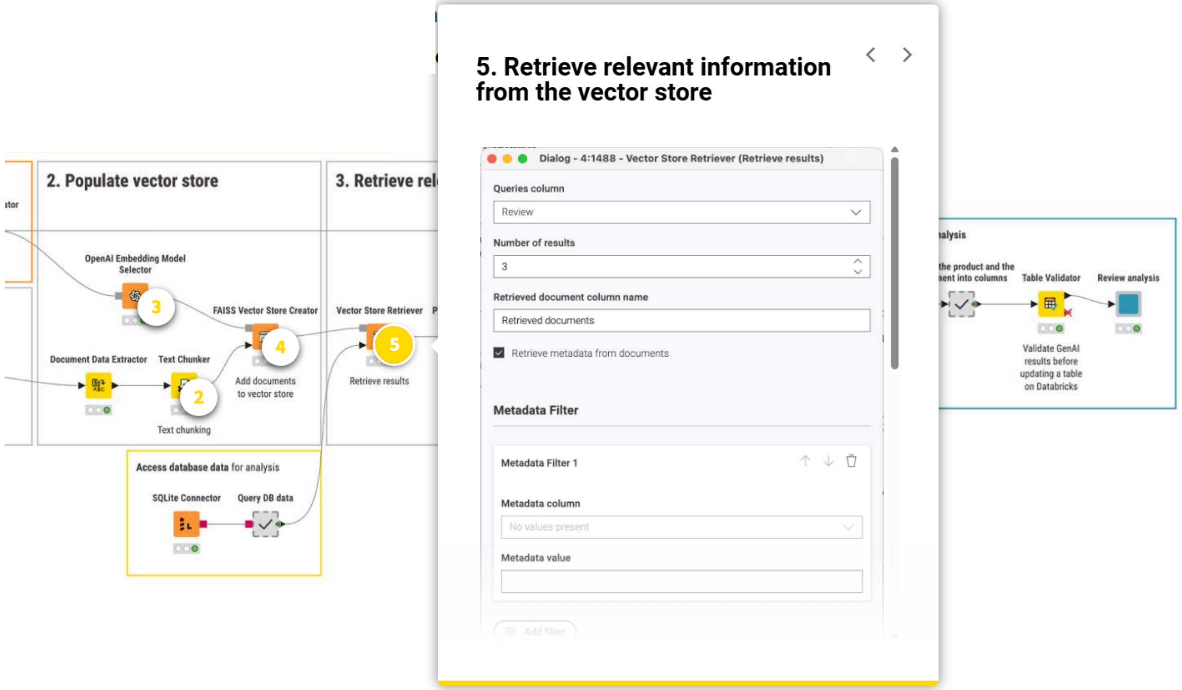
4- Create Vector Store

- Depending on the vector store (e.g., FAISS, Chroma), use the appropriate **Vector Store Creator** node to convert each text chunk into a high-dimensional vector with the selected embedding model.
- The original text data will also be stored, and optionally, we can include relevant metadata to help refine retrieval during later queries.



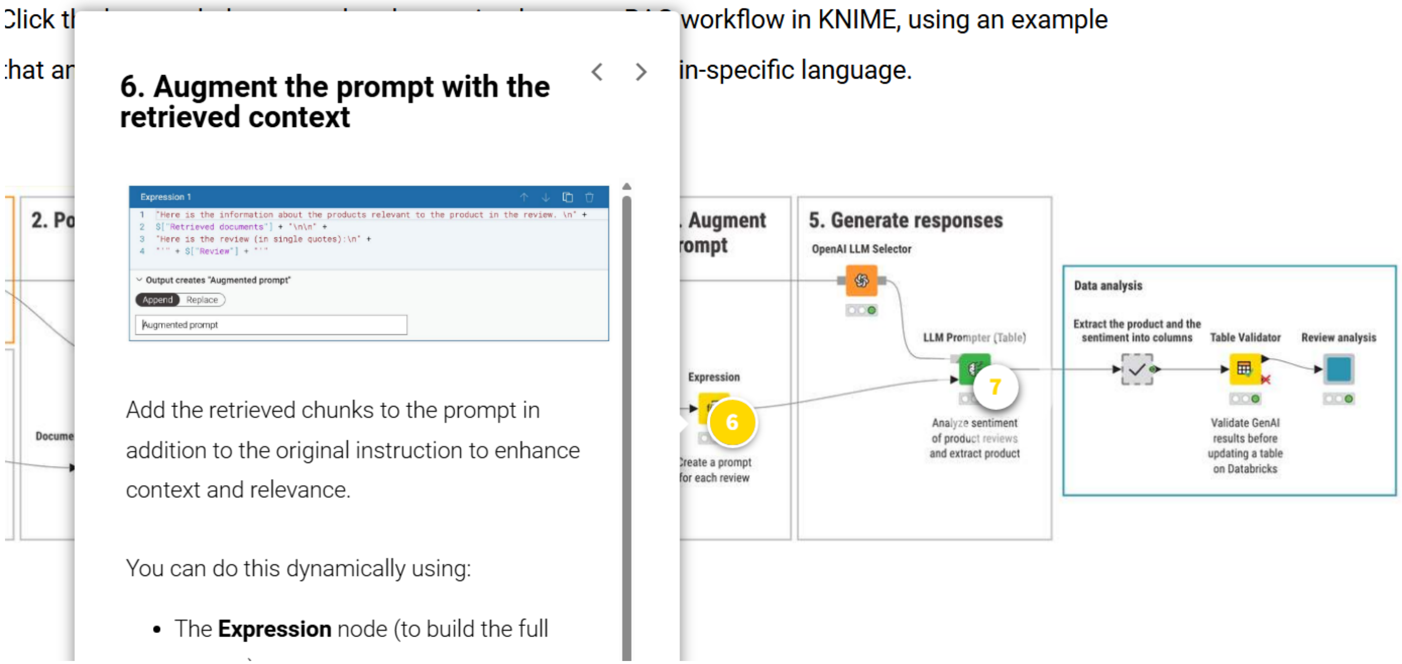
5- Retrieve relevant content from the vector store

- For a query that will be part of the prompt (or the initial, non-augmented prompt), retrieve the most relevant chunks from the knowledge base automatically using vector-based similarity search.
- Use the **Vector Store Retriever** node, regardless of the vector store used. We can:
    - We can specify how many documents to extract—they will be returned as a list.
    - Apply filters to narrow results based on metadata



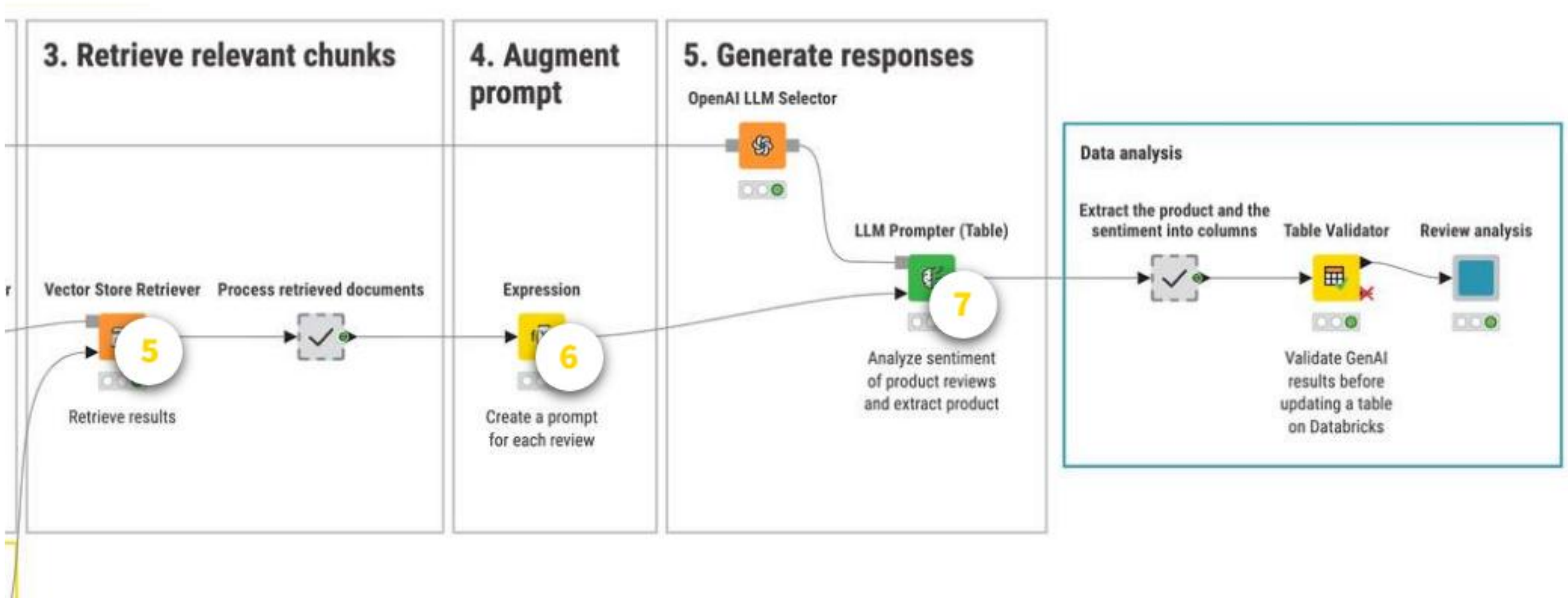
6- Augment the prompt

- Add the retrieved chunks to the prompt in addition to the original instruction to enhance context and relevance.
- We can do this dynamically using:
  - The **Expression** node (to build the full prompt),
  - The **Message Creator** node (to combine original prompt and retrieved chunks).

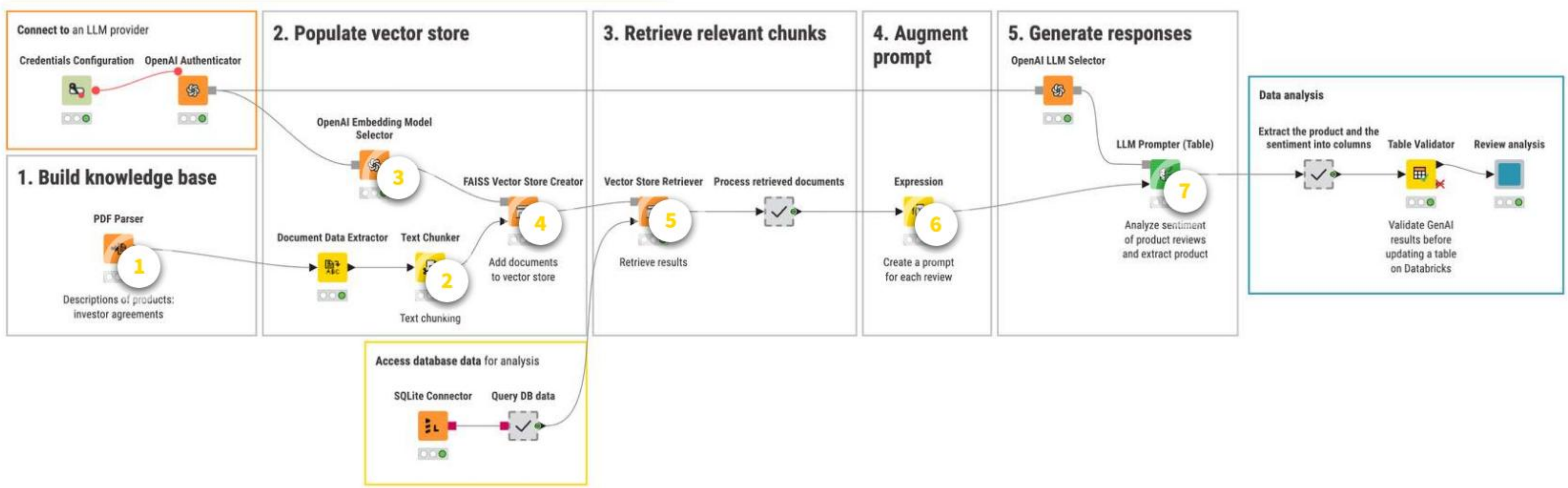


7- Submit the augmented prompt(s) to an LLM

- Send the augmented prompt as usual to the LLM and receive more tailored responses, enriched by the additional knowledge.



RAG Example in KNIME



This workflow can be downloaded as following:

1. Download Course Workflows from VClass
2. Goto Generative AI Folder -> AI Extension Guide -> RAG
3. Open Product FAQ